PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-222228
(43)Date of publication of application : 11.08.2000

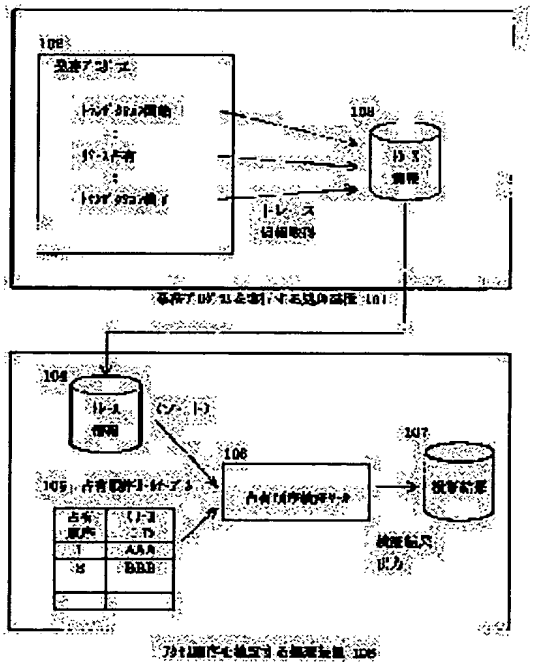
(51)Int.Cl. G06F 9/46
G06F 11/30
G06F 15/00

(21)Application number : 11-021249
(22)Date of filing : 29.01.1999

(71)Applicant : HITACHI LTD
(72)Inventor : IIJIMA MINORU

(54) DEADLOCK PREVENTING METHOD BY VERIFICATION OF RESOURCE OCCUPATION ORDER

(57)Abstract:
PROBLEM TO BE SOLVED: To verify the access order of resources in a test process and to pre vent a deadlock in actual operation by tracing the history of resource occupation and inputting the result to an access order verification tool of differently generat ed resources.
SOLUTION: A device comprises a processor 101 which conducts a system test of an operation program and a processor 108 which verifies access order. The operation program 102 codes a process for obtaining trace information 103 right after a process for occupying a resource. The processor 108 which verifies occupation order inputs trace information 104 and an occupation order rule table 105 to the occupation order verifying tool 106 to obtain a verification result 107. The occupation order of resources can be confirmed and order illegality can mechanically be detected. Further, the resource access order verifying tool can be generated independently of a program main body and exerts no influence on the development process of the program main body.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－222228
（Ｐ２０００－２２２２２８Ａ）
(43)公開日　平成12年８月11日(2000.8.11)

(51)Int.Cl.[7]	識別記号	ＦＩ			テーマコード（参考）
G06F 9/46	340	G06F 9/46	340	G	5B042
11/30	305	11/30	305	G	5B085
15/00	320	15/00	320	K	5B098

審査請求　未請求　請求項の数１　ＯＬ　（全５頁）

(21)出願番号　特願平11－21249

(22)出願日　平成11年１月29日(1999.1.29)

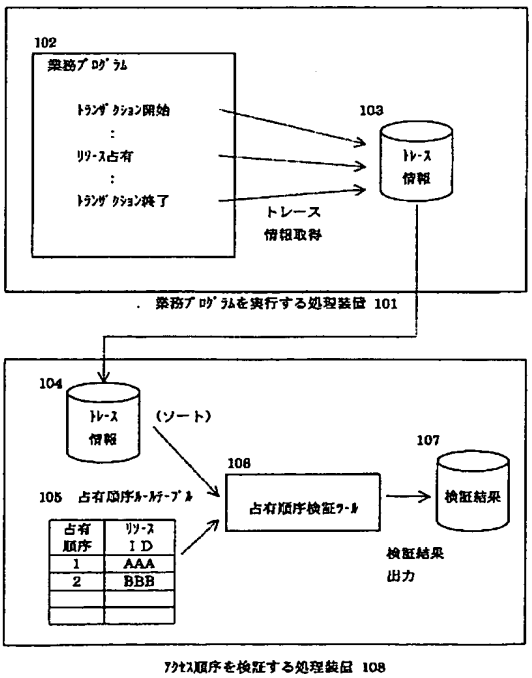
(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地
(72)発明者　飯島　実
神奈川県川崎市幸区鹿島田890番地　株式会社日立製作所情報システム事業部内
(74)代理人　100068504
弁理士　小川　勝男

Ｆターム(参考)　5B042 GA23 GB01 HH20 HH30 MA14
MC17 MC22 NN01
5B085 AC08 BA07
5B098 GA04 GB01 GD01 GD16 GD24
JJ07

(54)【発明の名称】リソース占有順序の検証によるデッドロック防止方法

(57)【要約】
【課題】オンラインシステムにおいては、資源アクセス順序の不正によりデッドロックが発生する可能性があるが、同時処理の一瞬のタイミングでしか発生しない場合が多いため、実機を使用してシステムテストを行っても必ずしもそれを検出できるとは限らない。
【解決手段】実機を使用したテストで取得したトレース情報から、リソースの占有順序を検証することにより、デッドロックの要因を摘出できる。

図１



1

【特許請求の範囲】

【請求項１】１トランザクションで多数のリソースへアクセスするマルチタスクのオンライントランザクション処理システムにおいて、実行時のトレースファイルからトランザクションが占有するリソースの占有順序規則を検証し、順序不正を摘出することにより、デッドロック要因の内在を事前に検知することを特徴とする排他制御におけるデッドロック防止方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明はオンライントランザクション処理システムのように１つのトランザクションで複数のリソースを占有要求するマルチタスクの環境に適用する。

【０００２】

【従来の技術】特開平７－１２１３８７号公報「排他制御におけるデッドロック防止方法」では実行時にＯＣＰ側でリソースアクセス順序をチェックする方式である。

【０００３】この方式は、データベースアクセスを含め、全てのリソースアクセスを１つのＯＣＰで管理していることが前提になる。

【０００４】近年、国際的標準化や流通パッケージソフトの普及により、ＯＣＰソフトやデータベースソフトはパッケージソフトとして一般に市販されているものを組み合わせて適用するのが普通であり、上記の方式は適用できない状況にある。

【０００５】業務プログラムの中でリソースアクセス順序をチェックするのは開発時の負担が大きく、本来の処理に対する悪影響が発生する可能性も少なくない。

【０００６】

【発明が解決しようとする課題】デッドロック防止のためにはリソース占有順序を決め、その順序を厳守して、プログラムを作成しなければならない。

【０００７】しかし、ヒューマンエラー等により、その順序が守られずに、デッドロック発生の原因になるという問題がある。

【０００８】本発明の目的は、テスト工程においてリソースのアクセス順序を検証し、実運用でのデッドロック発生を防止することである。

【０００９】

【課題を解決するための手段】オンラインシステムのプログラムではテスト結果の確認や障害発生時の調査を容易にするためにトレース情報をファイルに出力するのが一般的である。

【００１０】この中に資源占有の履歴をトレースし、別に作成したリソースのアクセス順序検証ツールの入力とすることにより、リソースアクセス順序不正を摘出することができる。

【００１１】

【発明の実施の形態】本発明の実施例を図面を参照して

2

説明する。

【００１２】まず、全体システム構成図を図１に示す。

【００１３】業務プログラムのシステムテストを行う処理装置１０１とアクセス順序を検証する処理装置１０８から構成される。業務プログラム１０２はリソースを占有する処理の直後にトレース情報１０３を取得する処理をコーディングしておく。占有順序を検証する処理装置１０８では、トレース情報１０４と占有順序ルールテーブル１０５を占有順序検証ツール１０６の入力として、検証結果１０７を得る。

【００１４】占有順序ルールテーブル１０５は図３に示すように、リソースＩＤ（302）ごとに占有順序（301）を規定したものである。

【００１５】次に、トレース取得情報の例を図４に示す。

【００１６】トランザクションＩＤ（401）はトランザクション毎に採番される識別子で、時刻（402）は占有要求の実時刻である。

【００１７】トレースファイルには同時に動作する複数のトランザクションのトレース情報が混合して出力される。占有順序の検証はトランザクションＩＤごとに行う必要があるため、検証の入力としてはトランザクションＩＤ（401）と時刻（402）をキーにしてソートしたものを使用する。

【００１８】ソート後はトランザクションＩＤ（411）ごとに検証単位（410）となる。

【００１９】モジュール名（403）、関数名（404）、行番号（405）は業務プログラム内のリソース占有個所を認識するための情報である。

【００２０】リソースＩＤ（406）は占有リソースを示す。トランザクションＩＤ（401）が同一のトレース情報について、このリソースＩＤ(406)は占有順序ルールテーブル１０５で規定された順序になっていることを検証しなければならない。

【００２１】次に、図２により、占有順序検証ツールの処理フロー例を説明する。

【００２２】（ステップ２０１）占有順序テーブルファイルをメモリ上のテーブルに読み込む。

【００２３】（ステップ２０２）占有順序番号の上昇順をチェックするための直前情報退避エリアを初期設定。

【００２４】（ステップ２０３）ソートされたトレースレコードを１件ずつ入力する。

【００２５】（ステップ２０４）トレースレコード終了まで繰り返す。

【００２６】（ステップ２０５）トレースレコード上のリソースＩＤ406から、占有順序ルールテーブル（302）をサーチし、占有順序番号（301）を検索し、トレース情報に付加する（図5　507）。

【００２７】（ステップ２０６）トランザクションＩＤが変わった場合、そのレコードは順序チェックしないの

でステップ209に分岐する。

【００２８】（ステップ２０７）占有順序番号の上昇順をチェックする。

【００２９】（ステップ２０８）上昇順でない場合、警告メッセージを付加する（図5　507）（ステップ２０９）占有順序番号507、警告メッセージ508を付加したトレース情報レコードを検証結果ファイルに出力する。

【００３０】（ステップ２１０）占有順序番号の上昇順をチェックするための直前情報退避エリアに今のトレース情報レコードを退避する。

【００３１】図５に検証結果の例を示す。ルールに違反しているアクセス部分、つまりルールではリソースＣＣＣ→ＤＤＤの順にアクセスすべきところを、その順序が逆になっているアクセスについては、占有順序番号（507）が上昇順になっていない。

【００３２】デッドロック要因として、警告メッセージ（508）を付加して出力する。

【００３３】

【発明の効果】本発明によれば、以下の効果が期待出来る。

【００３４】（１）リソースの占有順序を確認できると共に、順序不正を機械的に検出できる。

【００３５】（２）リソースアクセス順序検証ツールはプログラム本体と独立して作成することが可能であり、プログラム本体の開発工程に影響を与えない。

【００３６】（３）性能・リソースの面でプログラム本体に負荷をかけないので性能を重視するシステムには最適なデッドロック防止方式である。

【図面の簡単な説明】

【図１】本発明の実施例のシステム構成図。

【図２】アクセス順序検証ツールのフローチャート。

【図３】アクセス順序ルールを記述したテーブル例を示す図。

【図４】テスト時に取得されるトレース情報の例を示す図。

【図５】アクセス順序検証ツールによる検証結果の例を示す図。

【符号の説明】

１０１…業務プログラムをシステムテストする処理装置、１０２…業務プログラム、１０３…テスト実行時に出力されるトレース情報、１０４…検証の入力となるトレース情報、１０５…占有順序ルールテーブル、１０６…占有順序検証ツール、１０７…検証結果出力ファイル、１０８…占有順序を検証する処理装置、３０１…占有順序ルールテーブル上の占有順序番号、３０２…占有順序ルールテーブル上のリソースＩＤ、４０１…トレース情報上のトランザクションＩＤ、４０２…トレース情報上の時刻、４０３…トレース情報上のモジュール名、４０４…トレース情報上の関数名、４０５…トレース情報上の行番号、４０６…トレース情報上のリソースＩＤ、４１０…トレース情報の検証単位、４１１…トレース情報上のトランザクションＩＤ（ソート後）、５０１…検証結果出力ファイル上のトランザクションＩＤ、５０６…検証結果出力ファイル上のリソースＩＤ、５０７…検証結果出力ファイル上の占有順序番号（検証ツールで付加）、５０８…検証結果出力ファイル上の警告メッセージ（検証ツールで付加）、５１０…検証結果出力ファイル上の検証単位。

【図３】

図３

301 占有順序番号	302 リソースＩＤ
1	ＡＡＡ
2	ＢＢＢ
3	ＣＣＣ
4	ＤＤＤ
:	:

【図５】

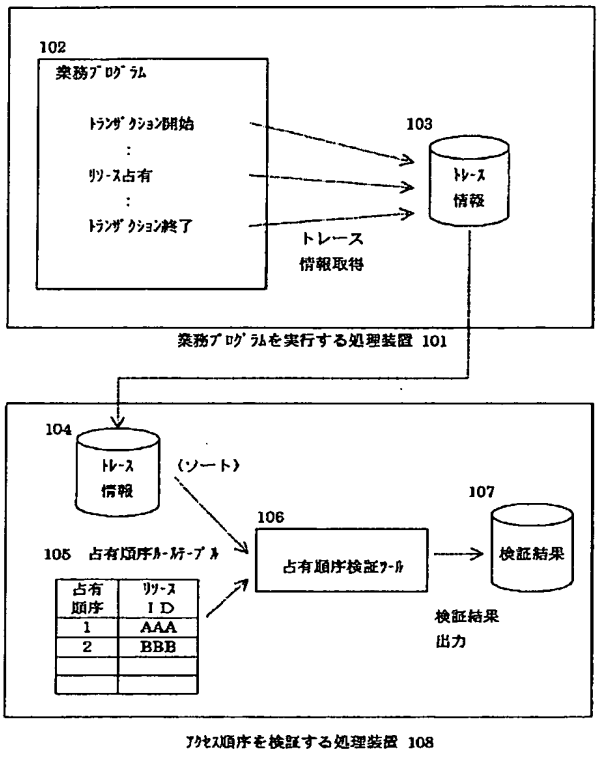
図５

501 トランザクションID	時刻	モジュール名	関数名	行番号	506 リソースID	507 占有順序番号	508 警告メッセージ	510
ttt0001	xxxxxx	XXXX	XXXX	nnnn	AAA	1		↑
	xxxxxx	XXXX	XXXX	nnnn	CCC	3		検証単位
	xxxxxx	XXXX	XXXX	nnnn	DDD	4		↓
ttt0002	xxxxxx	XXXX	XXXX	nnnn	AAA	1		↑
	xxxxxx	XXXX	XXXX	nnnn	DDD	4		検証単位
	xxxxxx	XXXX	XXXX	nnnn	CCC	3	←警告	↓
ttt0003	xxxxxx	XXXX	XXXX	nnnn	BBB	2		

同じトランザクションＩＤで
占有順序番号が上昇順でない

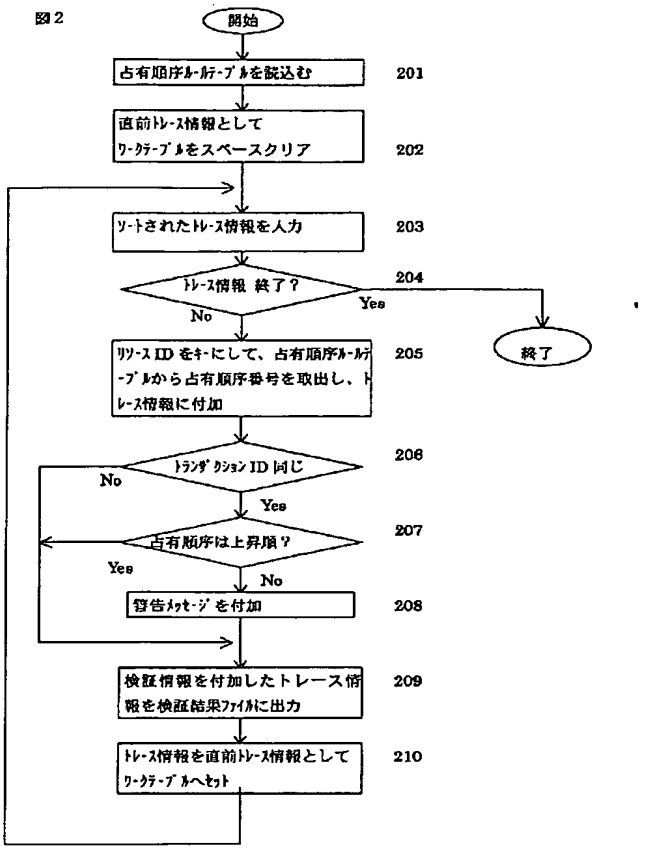
【図１】

図１


102
業務プログラム
トランザクション開始
：
リソース占有
：
トランザクション終了
103
トレース情報
トレース情報取得
業務プログラムを実行する処理装置 101
104
トレース情報
（ソート）
105 占有順序ルールテーブル
占有順序
リソースID
1
AAA
2
BBB
106
占有順序検証ツール
107
検証結果
検証結果出力
アクセス順序を検証する処理装置 108

【図２】

図２


開始
占有順序ルールテーブルを読込む
201
直前トレース情報としてワークテーブルをスペースクリア
202
ソートされたトレース情報を入力
203
トレース情報 終了？
204
No
Yes
終了
リソースIDをキーにして、占有順序ルールテーブルから占有順序番号を取出し、トレース情報に付加
205
トランザクションID同じ
206
No
Yes
占有順序は上昇順？
207
Yes
No
警告メッセージを付加
208
検証情報を付加したトレース情報を検証結果ファイルに出力
209
トレース情報を直前トレース情報としてワークテーブルへセット
210

【図４】

図４

（ａ）トレース取得時

401 トランザクション ID	402 時刻	403 モジュール名	404 関数名	405 行番号	406 リソース ＩＤ
tttt0001	xxxxxx	XXXX	XXXX	nnnn	AAA
tttt0002	xxxxxx	XXXX	XXXX	nnnn	AAA
tttt0003	xxxxxx	XXXX	XXXX	nnnn	BBB
tttt0002	xxxxxx	XXXX	XXXX	nnnn	DDD
tttt0001	xxxxxx	XXXX	XXXX	nnnn	CCC
tttt0001	xxxxxx	XXXX	XXXX	nnnn	DDD
tttt0002	xxxxxx	XXXX	XXXX	nnnn	CCC

（ｂ）検証のためのソート後

411 トランザクション ID	時刻	モジュール名	関数名	行番号	リソース ＩＤ	410
tttt0001	xxxxxx	XXXX	XXXX	nnnn	AAA	↑
tttt0001	xxxxxx	XXXX	XXXX	nnnn	CCC	検証単位
tttt0001	xxxxxx	XXXX	XXXX	nnnn	DDD	↓
tttt0002	xxxxxx	XXXX	XXXX	nnnn	AAA	↑
tttt0002	xxxxxx	XXXX	XXXX	nnnn	DDD	検証単位
tttt0002	xxxxxx	XXXX	XXXX	nnnn	CCC	↓
tttt0003	xxxxxx	XXXX	XXXX	nnnn	BBB	

↑
トランザクションＩＤごとに
時系列な並びにソート